AB32开发板笔记笔记（避坑手册）ByChow

开发板AB32VG1资料

- 中科蓝讯AB32VG1开发实践指南

```
https://file.elecfans.com/web2/M00/14/6F/pYYBAGE-y4WAGoojANBtfl0No2g719.pdf
```

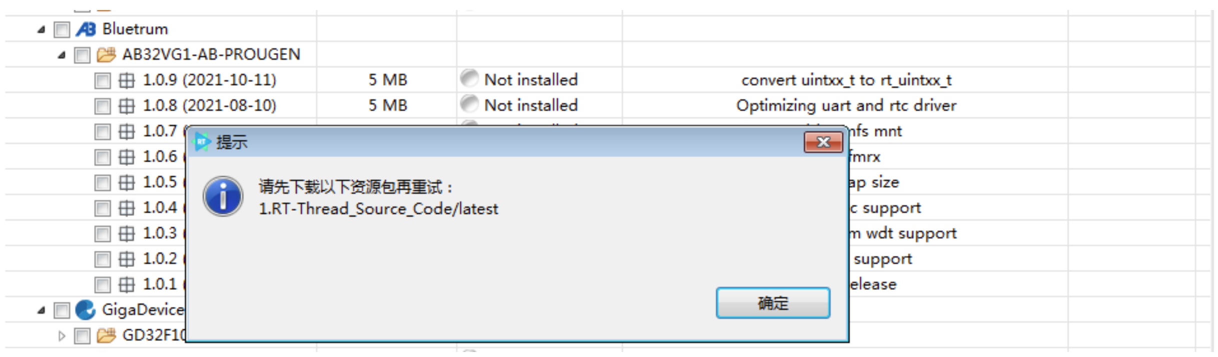
系统环境说明 (Windows 7)

- 作者:duxingkei
- 邮箱：277563381@qq.com
- ▼ 开发环境清单列表
  - WIN7 64Bit
  - ▼ RT-Thread Studio(202108311200)

    版本: 2.1.2
    构建ID: 202108311200

AB32VG必须使用最新latest版本的rtt，否则报错

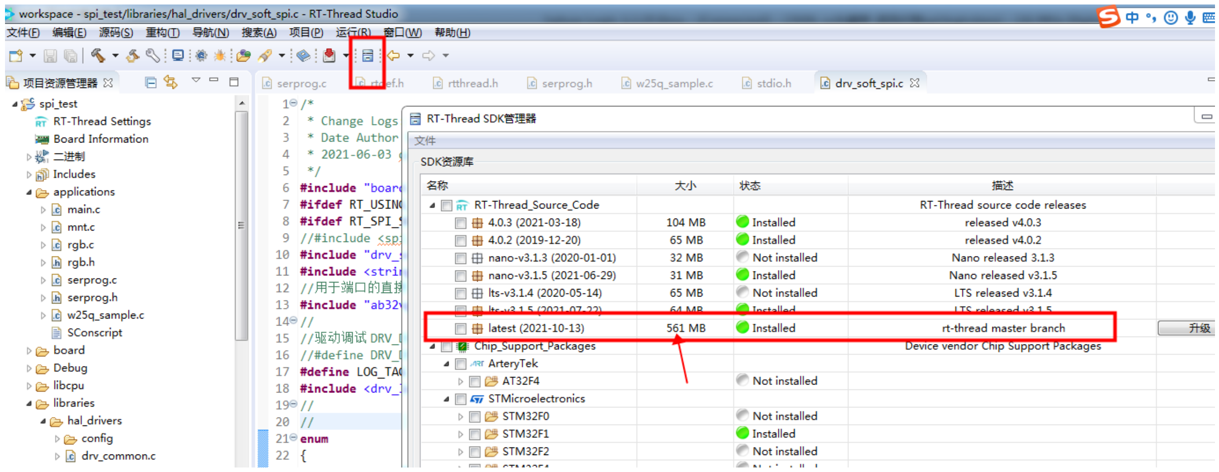
- 报错图



```
Bluetrum
  AB32VG1-AB-PROUGEN
    1.0.9 (2021-10-11)   5 MB   Not installed   convert uintxx_t to rt_uintxx_t
    1.0.8 (2021-08-10)   5 MB   Not installed   Optimizing uart and rtc driver
    1.0.7                                       nfs mnt
    1.0.6                                       fmrx
    1.0.5                                       ap size
    1.0.4                                       c support
    1.0.3                                       m wdt support
    1.0.2                                       support
    1.0.1                                       elease
GigaDevice
  GD32F1
```

```
提示
请先下载以下资源包再重试：
1.RT-Thread_Source_Code/latest
确定
```

- 必须下载最大的那个rtt包安装（latest 561M）



RT-Thread SDK管理器

SDK资源库

名称	大小	状态	描述
RT-Thread_Source_Code			RT-Thread source code releases
4.0.3 (2021-03-18)	104 MB	Installed	released v4.0.3
4.0.2 (2019-12-20)	65 MB	Installed	released v4.0.2
nano-v3.1.3 (2020-01-01)	32 MB	Not installed	Nano released 3.1.3
nano-v3.1.5 (2021-06-29)	31 MB	Installed	Nano released v3.1.5
lts-v3.1.4 (2020-05-14)	65 MB	Not installed	LTS released v3.1.4
lts-v3.1.5 (2021-07-22)	64 MB	Installed	LTS released v3.1.5
latest (2021-10-13)	561 MB	Installed	rt-thread master branch
Chip_Support_Packages			Device vendor Chip Support Packages
ArteryTek			
AT32F4		Not installed	
STMicroelectronics			
STM32F0		Not installed	
STM32F1		Installed	
STM32F2		Not installed	

-

用AB32VG开发板的spi模拟方式操作读写挂载格式化等，巨大的坑，速度超慢

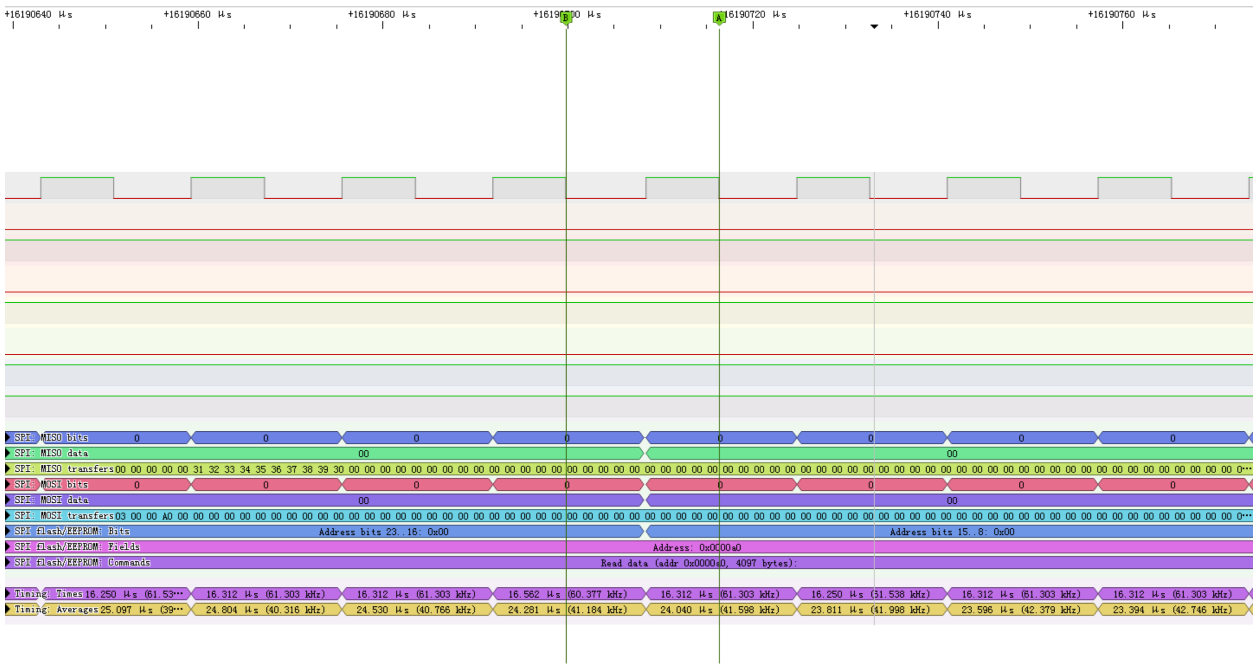
- 代码全部来自 《中科蓝讯AB32VG1开发套件.pdf》
- 原始代码采用的spi模拟延时 是根据RTT的tick数量来延时的，最快实测也就330Hz；
- 修改原始代码 `drv_soft_spi.c` 的延时函数 `spi_delay` ,不做延时,直接退出，实测达到最大60kHz,速度超快

```
static inline void spi_delay(rt_uint32_t us)
{
    if(us==0)return;
    //延时非US级别,而是1个tick的周期,传入0,spi实测频率为330hz,超级慢
  //  rt_thread_mdelay(us);
}
```

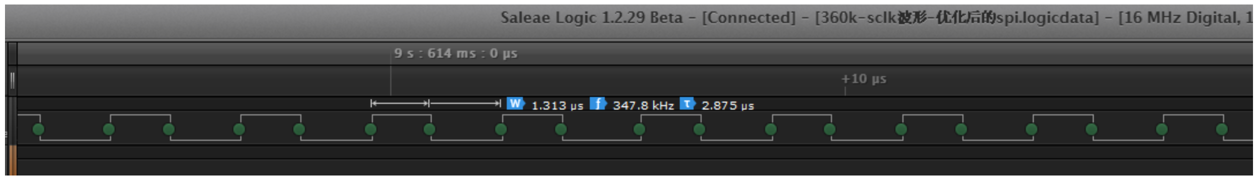
- 屏蔽了 `drv_soft_spi.c` 里面的spi调试代码宏开关更快

```
//驱动调试 DRV_DEBUG 不定义则无此调试信息输出
//#define DRV_DEBUG
```

修复AB32VG模拟spi速度超慢bug重新操作，速度超快



让SPI速度更快点，直接屏蔽spi的delay，端口操作不在用pin驱动，直接IO读写，速度从60kHz提高到360kHz(sclk)


Saleae Logic 1.2.29 Beta – [Connected] – [360k-sclk波形-优化后的spi.logicdata] – [16 MHz Digital, 1
9 s : 614 ms : 0 μs
+10 μs
W 1.313 μs
f 347.8 kHz
T 2.875 μs

▼ 参考代码

```
/*
 * Change Logs:
 * Date Author Notes
 * 2021-06-03 qwz first version
 */
#include "board.h"
#ifdef RT_USING_SPI
#ifdef RT_SPI_SOFT
//#include <spi.h>
#include "drv_soft_spi.h"
#include <string.h>
//用于端口的直接读写,加速控制
#include "ab32vg1_hal.h"
//
//驱动调试 DRV_DEBUG 不定义则无此调试信息输出
//#define DRV_DEBUG
#define LOG_TAG "drv.spisoft"
#include <drv_log.h>
//
//
enum
{
#ifdef BSP_USING_SOFT_SPI1
    SOFT_SPI1_INDEX,
#endif
};

#define SPI0_BUS "spi0"
```

```
//PIN MAP
// incording drv_gpio.h
/*
 #define __AB32_PORT(port)   GPIO##port
 #define __AB32_GET_PIN_A(PIN)  PIN
 #define __AB32_GET_PIN_B(PIN)  8 + PIN
 #define __AB32_GET_PIN_E(PIN)  13 + PIN
 #define __AB32_GET_PIN_F(PIN)  21 + PIN
 */
//0-----8 9   10   11  12  13  14   15  16  17  18  19  20   21  22  23  24  25  26  27  28
//PA0-PA8 PB0 PB1  PB2 PB3 PE0 PE1  PE2 PE3 PE4 PE5 PE6 PE7  PF0 PF1 PF2 PF3 PF4 PF5 PF6 PF7
//PB2 10 ;PE5 18;PE6 19;PB1 9;
/*
 #define SPI0_MOSI_PIN "PE.5"
 #define SPI0_MISO_PIN "PB.2"
 #define SPI0_SCLK_PIN "PB.1"

 #define SOFT_SPI1_BUS_CONFIG { \
.mosi_pin = 18, \
.miso_pin = 10, \
.sclk_pin = 9, \
.bus_name = "spi0", \
}
 */
//
#define SPI0_MOSI_PIN "PE.1"
#define SPI0_MISO_PIN "PF.0"
#define SPI0_SCLK_PIN "PE.0"
//CS = PA.5
#define SOFT_SPI1_BUS_CONFIG { \
.mosi_pin = 14, \
.miso_pin = 21, \
.sclk_pin = 13, \
.bus_name = "spi0", \
}

#if 1

static struct ab32_soft_spi_config soft_spi_config[] = {
#ifdef BSP_USING_SOFT_SPI1
        SOFT_SPI1_BUS_CONFIG,
#endif
        };

#define spi_mosi_out_1() hal_gpio_write(GPIOE_BASE, 1, GPIO_PIN_HIGH)
#define spi_mosi_out_0() hal_gpio_write(GPIOE_BASE, 1, GPIO_PIN_LOW)

#define spi_miso_out_1() hal_gpio_write(GPIOF_BASE, 0, GPIO_PIN_HIGH)
#define spi_miso_out_0() hal_gpio_write(GPIOF_BASE, 0, GPIO_PIN_LOW)

#define spi_sclk_out_1() hal_gpio_write(GPIOE_BASE, 0, GPIO_PIN_HIGH)
#define spi_sclk_out_0() hal_gpio_write(GPIOE_BASE, 0, GPIO_PIN_LOW)

#define spi_read_mosi() hal_gpio_read(GPIOE_BASE, 1)
#define spi_read_miso() hal_gpio_read(GPIOF_BASE,0)
#define spi_read_sclk() hal_gpio_read(GPIOE_BASE,0)

static struct ab32_soft_spi soft_spi_bus_obj[sizeof(soft_spi_config) / sizeof(soft_spi_config[0])] = { 0 };
static rt_err_t ab32_spi_init(struct ab32_soft_spi *spi_drv, struct rt_spi_configuration *cfg)
{
    RT_ASSERT(spi_drv != RT_NULL);RT_ASSERT(cfg != RT_NULL);

//mode = master
    if (cfg->mode & RT_SPI_SLAVE)
    {
        return RT_EIO;
    }
    else
    {
        spi_drv->mode = RT_SPI_MASTER;
    }
    if (cfg->mode & RT_SPI_3WIRE)
    {
        return RT_EIO;
    }
    if (cfg->data_width == 8 || cfg->data_width == 16)
        spi_drv->data_width = cfg->data_width;
    else
    {
        return RT_EIO;
    }
    if (cfg->mode & RT_SPI_CPHA)
    {
        spi_drv->cpha = 1;
    }
    else
    {
        spi_drv->cpha = 0;
    }
    if (cfg->mode & RT_SPI_CPOL)
    {
        spi_drv->cpol = 1;
    }
    else
    {
        spi_drv->cpol = 0;
    }
    if (cfg->mode & RT_SPI_NO_CS)
    {
    }
    else
    {
    }
    if (cfg->max_hz >= 1200000)
    {
        spi_drv->spi_delay = 0;

    }
    else if (cfg->max_hz >= 1000000)
    {
        spi_drv->spi_delay = 8;
    }
    else if (cfg->max_hz >= 830000)
    {
        spi_drv->spi_delay = 16;
    }
    else
    {
        spi_drv->spi_delay = 24;
    }LOG_D("SPI limiting freq: %d, BaudRatePrescaler: %d,spi_drv->spi_delay: %d",
            cfg->max_hz,
            spi_drv->max_hz,spi_drv->spi_delay);
    if (cfg->mode & RT_SPI_MSB)
    {
        spi_drv->msb = 1;
    }
    else
    {
        spi_drv->msb = 0;
    }
    rt_pin_mode(spi_drv->config->mosi_pin, PIN_MODE_OUTPUT_OD);
    rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
    rt_pin_mode(spi_drv->config->miso_pin, PIN_MODE_INPUT_PULLDOWN);
    rt_pin_mode(spi_drv->config->sclk_pin, PIN_MODE_OUTPUT_OD);
    if (spi_drv->cpol)
        rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
    else
        rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
    LOG_D("%s init done", spi_drv->config->bus_name);
    return RT_EOK;
}
static inline void spi_delay(rt_uint32_t us)
{
    //  if(us==0)return;
    //延时非US级别,而是1个tick的周期,传入0,spi实测频率为330hz,超级慢
    //  rt_thread_mdelay(us);
}
static rt_uint32_t soft_spi_read_write_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* send_buff,
        rt_uint8_t* recv_buff, rt_uint32_t len)
{
    rt_uint8_t dataIndex = 0;
    rt_uint8_t time = 1;
    for (rt_uint32_t i = 0; i < len; i++)
    {
        if (spi_drv->cpha)
        { //CPHA=1
          //if (rt_pin_read(spi_drv->config->sclk_pin))
            if (spi_read_sclk() == 1)
            {
                //rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                spi_sclk_out_0();
            }
            else
            {
                //rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                spi_sclk_out_1();

            }
        }
        if (spi_drv->data_width == 16)
            time = 2;
        do
        {
            for (rt_uint8_t j = 0; j < 8; j++)
            {
                if ((send_buff[dataIndex] & 0x80) != 0)
                {
                    // rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
                    spi_mosi_out_1();
                }
                else
                {
                    // rt_pin_write(spi_drv->config->mosi_pin, PIN_LOW);
                    spi_mosi_out_0();
                }
                send_buff[dataIndex] <<= 1;
                //spi_delay(spi_drv->spi_delay);
                // if (rt_pin_read(spi_drv->config->sclk_pin))
                if (spi_read_sclk() == 1)
                {
                    //rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    spi_sclk_out_0();
                }
                else
                {
                    //rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    spi_sclk_out_1();
                }
                recv_buff[dataIndex] <<= 1;
                //if (rt_pin_read(spi_drv->config->miso_pin))
                if (spi_read_miso())
                {
                    recv_buff[dataIndex] |= 0x01;
                }
                //spi_delay(spi_drv->spi_delay);
                if (time != 0 || j != 7)
                {
                    //if (rt_pin_read(spi_drv->config->sclk_pin))
                    if (spi_read_sclk() == 1)
                    {
                        //rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                        spi_sclk_out_0();
                    }
                    else
                    {
                        //rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                        spi_sclk_out_1();
                    }
                }
            }
            dataIndex++;
        } while ((--time) == 1);
        time = 1;
        // spi_delay(spi_drv->spi_delay);
    }
    return len;
}
static rt_uint32_t soft_spi_read_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* recv_buff, rt_uint32_t len)
{
    rt_uint8_t send_buff = spi_drv->dummy_data;
    rt_uint32_t dataIndex = 0;
    rt_uint8_t time = 1;
    if (spi_drv->cpha)
    { //CPHA=1
      // if (rt_pin_read(spi_drv->config->sclk_pin))
        if (spi_read_sclk() == 1)
        {
            // rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
            spi_sclk_out_0();
        }
        else
        {
            // rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
            spi_sclk_out_1();
        }
    }
    for (rt_uint32_t i = 0; i < len; i++)
    {
        if (spi_drv->data_width == 16)
            time = 2;
        do
        {
            for (rt_uint8_t j = 0; j < 8; j++)
            {
                if ((send_buff & 0x80) != 0)
                {
                    // rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
                    spi_mosi_out_1();
                }
                else
                {
                    //rt_pin_write(spi_drv->config->mosi_pin, PIN_LOW);
                    spi_mosi_out_0();
                }
                send_buff <<= 1;
                //spi_delay(spi_drv->spi_delay);
                //if (rt_pin_read(spi_drv->config->sclk_pin))
                if (spi_read_sclk() == 1)
                {
                    //rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    spi_sclk_out_0();
                }
                else
                {
                    //rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    spi_sclk_out_1();
                }
                *recv_buff <<= 1;
                // if (rt_pin_read(spi_drv->config->miso_pin))
                if (spi_read_miso() == 1)
                {
                    *recv_buff |= 0x01;
                }
                else
                {
                    *recv_buff &= 0xfe;
                }
                //spi_delay(spi_drv->spi_delay);
                if (time != 0 || j != 7)
                {
                    //if (rt_pin_read(spi_drv->config->sclk_pin))
                    if (spi_read_sclk() == 1)
                    {
                        // rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                        spi_sclk_out_0();
                    }
                    else
                    {
                        // rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                        spi_sclk_out_1();
                    }
                }
            }
            recv_buff++;
            dataIndex++;
        } while ((--time) == 1);
        time = 1;
        // spi_delay(spi_drv->spi_delay);
        LOG_D("DONE ONE BYTE %d",dataIndex);LOG_D("%d",spi_drv->spi_delay);
        //LOG_I("DONE ONE BYTE %d",dataIndex); LOG_D("%d",spi_drv->spi_delay);
    }
    return len;
}
static rt_uint32_t soft_spi_write_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* send_buff, rt_uint32_t len)
{
    rt_uint8_t recv_buff = 0;
    rt_uint32_t dataIndex = 0;
    rt_uint8_t time = 1;
    LOG_D("%x",send_buff[0]);
    if (spi_drv->cpha)
    { //CPHA=1
#if 1
        //if (rt_pin_read(spi_drv->config->sclk_pin))
        if (spi_read_sclk() == 1)
        {
            // rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
            spi_sclk_out_0();
        }
        else
        {
            // rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
            spi_sclk_out_1();
        }
#else
        if (rt_pin_read(spi_drv->config->sclk_pin))
        {
            rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
        }
        else
        {
            rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
        }
#endif
    }
    for (uint32_t i = 0; i < len; i++)
    {
        if (spi_drv->data_width == 16)
            time = 2;
        do
        {
            for (rt_uint8_t j = 0; j < 8; j++)
            {
                if ((send_buff[dataIndex] & 0x80) != 0)
                {
                    // rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
                    spi_mosi_out_1();
                    LOG_D("PIN_HIGH");
                }
                else
                {
                    //rt_pin_write(spi_drv->config->mosi_pin, PIN_LOW);
                    spi_mosi_out_0();
                    LOG_D("PIN_LOW");
                }
                send_buff[dataIndex] <<= 1;
                //spi_delay(spi_drv->spi_delay);
#if 1
                //if (rt_pin_read(spi_drv->config->sclk_pin))
                if (spi_read_sclk() == 1)
                {
                    // rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    spi_sclk_out_0();
                }
                else
                {
                    // rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    spi_sclk_out_1();
                }
#else
                if (rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                }
                else
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                }
#endif
                recv_buff <<= 1;
                // if (rt_pin_read(spi_drv->config->miso_pin))
                if (spi_read_miso() == 1)
                    recv_buff |= 0x01;
                //spi_delay(spi_drv->spi_delay);
                if (time != 0 || j != 7)
                {
#if 1
                    //if (rt_pin_read(spi_drv->config->sclk_pin))
                    if (spi_read_sclk() == 1)
                    {
                        // rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                        spi_sclk_out_0();
                    }
                    else
                    {
                        // rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                        spi_sclk_out_1();
                    }
#else
                    if (rt_pin_read(spi_drv->config->sclk_pin))
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    }
                    else
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    }
#endif
                }
            }
            dataIndex++;
        } while ((--time) == 1);
        time = 1;
      //  spi_delay(spi_drv->spi_delay);
    }
    return len;
}
#else

static struct ab32_soft_spi_config soft_spi_config[] =
{
#ifdef BSP_USING_SOFT_SPI1
    SOFT_SPI1_BUS_CONFIG,
#endif
};

static struct ab32_soft_spi soft_spi_bus_obj[sizeof(soft_spi_config) / sizeof(soft_spi_config[0])] =
{   0};
static rt_err_t ab32_spi_init(struct ab32_soft_spi *spi_drv, struct rt_spi_configuration *cfg)
{
    RT_ASSERT(spi_drv != RT_NULL);RT_ASSERT(cfg != RT_NULL);
//mode = master
    if (cfg->mode & RT_SPI_SLAVE)
    {
        return RT_EIO;
    }
    else
    {
        spi_drv->mode = RT_SPI_MASTER;
    }
    if (cfg->mode & RT_SPI_3WIRE)
    {
        return RT_EIO;
    }
    if (cfg->data_width == 8 || cfg->data_width == 16)
    spi_drv->data_width = cfg->data_width;
    else
    {
        return RT_EIO;
    }
    if (cfg->mode & RT_SPI_CPHA)
    {
        spi_drv->cpha = 1;
    }
    else
    {
        spi_drv->cpha = 0;
    }
    if (cfg->mode & RT_SPI_CPOL)
    {
        spi_drv->cpol = 1;
    }
    else
    {
        spi_drv->cpol = 0;
    }
    if (cfg->mode & RT_SPI_NO_CS)
    {
    }
    else
    {
    }
    if (cfg->max_hz >= 1200000)
    {
        spi_drv->spi_delay = 0;

    }
    else if (cfg->max_hz >= 1000000)
    {
        spi_drv->spi_delay = 8;
    }
    else if (cfg->max_hz >= 830000)
    {
        spi_drv->spi_delay = 16;
    }
    else
    {
        spi_drv->spi_delay = 24;
    }LOG_D("SPI limiting freq: %d, BaudRatePrescaler: %d,spi_drv->spi_delay: %d",
            cfg->max_hz,
            spi_drv->max_hz,spi_drv->spi_delay);
    if (cfg->mode & RT_SPI_MSB)
    {
        spi_drv->msb = 1;
    }
    else
    {
        spi_drv->msb = 0;
    }
    rt_pin_mode(spi_drv->config->mosi_pin, PIN_MODE_OUTPUT_OD);
    rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
    rt_pin_mode(spi_drv->config->miso_pin, PIN_MODE_INPUT_PULLDOWN);
    rt_pin_mode(spi_drv->config->sclk_pin, PIN_MODE_OUTPUT_OD);
    if (spi_drv->cpol)
    rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
    else
    rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
    LOG_D("%s init done", spi_drv->config->bus_name);
    return RT_EOK;
}
static inline void spi_delay(rt_uint32_t us)
{
    //  if(us==0)return;
    //延时非US级别,而是1个tick的周期,传入0,spi实测频率为330hz,超级慢
    //  rt_thread_mdelay(us);
}
static rt_uint32_t soft_spi_read_write_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* send_buff,
        rt_uint8_t* recv_buff, rt_uint32_t len)
{
    rt_uint8_t dataIndex = 0;
    rt_uint8_t time = 1;
    for (rt_uint32_t i = 0; i < len; i++)
    {
        if (spi_drv->cpha)
        { //CPHA=1
            if (rt_pin_read(spi_drv->config->sclk_pin))
            {
                rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
            }
            else
            {
                rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
            }
        }
        if (spi_drv->data_width == 16)
        time = 2;
        do
        {
            for (rt_uint8_t j = 0; j < 8; j++)
            {
                if ((send_buff[dataIndex] & 0x80) != 0)
                {
                    rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
                }
                else
                {
                    rt_pin_write(spi_drv->config->mosi_pin, PIN_LOW);
                }
                send_buff[dataIndex] <<= 1;
                spi_delay(spi_drv->spi_delay);
                if (rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                }
                else
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                }
                recv_buff[dataIndex] <<= 1;
                if (rt_pin_read(spi_drv->config->miso_pin))
                recv_buff[dataIndex] |= 0x01;
                spi_delay(spi_drv->spi_delay);
                if (time != 0 || j != 7)
                {
                    if (rt_pin_read(spi_drv->config->sclk_pin))
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    }
                    else
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    }
                }
            }
            dataIndex++;
        }while ((--time) == 1);
        time = 1;
        spi_delay(spi_drv->spi_delay);
    }
    return len;
}
static rt_uint32_t soft_spi_read_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* recv_buff, rt_uint32_t len)
{
    rt_uint8_t send_buff = spi_drv->dummy_data;
    rt_uint32_t dataIndex = 0;
    rt_uint8_t time = 1;
    if (spi_drv->cpha)
    { //CPHA=1
        if (rt_pin_read(spi_drv->config->sclk_pin))
        {
            rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
        }
        else
        {
            rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
        }
    }
    for (rt_uint32_t i = 0; i < len; i++)
    {
        if (spi_drv->data_width == 16)
        time = 2;
        do
        {
            for (rt_uint8_t j = 0; j < 8; j++)
            {
                if ((send_buff & 0x80) != 0)
                {
                    rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
                }
                else
                {
                    rt_pin_write(spi_drv->config->mosi_pin, PIN_LOW);
                }
                send_buff <<= 1;
                spi_delay(spi_drv->spi_delay);
                if (rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                }
                else
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                }
                *recv_buff <<= 1;
                if (rt_pin_read(spi_drv->config->miso_pin))
                {
                    *recv_buff |= 0x01;
                }
                else
                {
                    *recv_buff &= 0xfe;
                }
                spi_delay(spi_drv->spi_delay);
                if (time != 0 || j != 7)
                {
                    if (rt_pin_read(spi_drv->config->sclk_pin))
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    }
                    else
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    }
                }
            }
            recv_buff++;
            dataIndex++;
        }while ((--time) == 1);
        time = 1;
        spi_delay(spi_drv->spi_delay);
        LOG_D("DONE ONE BYTE %d",dataIndex);LOG_D("%d",spi_drv->spi_delay);
        //LOG_I("DONE ONE BYTE %d",dataIndex); LOG_D("%d",spi_drv->spi_delay);
    }
    return len;
}
static rt_uint32_t soft_spi_write_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* send_buff, rt_uint32_t len)
{
    rt_uint8_t recv_buff = 0;
    rt_uint32_t dataIndex = 0;
    rt_uint8_t time = 1;
    LOG_D("%x",send_buff[0]);
    if (spi_drv->cpha)
    { //CPHA=1
        if (rt_pin_read(spi_drv->config->sclk_pin))
        {
            rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
        }
        else
        {
            rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
        }
    }
    for (uint32_t i = 0; i < len; i++)
    {
        if (spi_drv->data_width == 16)
        time = 2;
        do
        {
            for (rt_uint8_t j = 0; j < 8; j++)
            {
                if ((send_buff[dataIndex] & 0x80) != 0)
                {
                    rt_pin_write(spi_drv->config->mosi_pin, PIN_HIGH);
                    LOG_D("PIN_HIGH");
                }
                else
                {
                    rt_pin_write(spi_drv->config->mosi_pin, PIN_LOW);
                    LOG_D("PIN_LOW");
                }
                send_buff[dataIndex] <<= 1;
                spi_delay(spi_drv->spi_delay);
                if (rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                }
                else
                {
                    rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                }
                recv_buff <<= 1;
                if (rt_pin_read(spi_drv->config->miso_pin))
                recv_buff |= 0x01;
                spi_delay(spi_drv->spi_delay);
                if (time != 0 || j != 7)
                {
                    if (rt_pin_read(spi_drv->config->sclk_pin))
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_LOW);
                    }
                    else
                    {
                        rt_pin_write(spi_drv->config->sclk_pin, PIN_HIGH);
                    }
                }
            }
            dataIndex++;
        }while ((--time) == 1);
        time = 1;
        spi_delay(spi_drv->spi_delay);
    }
    return len;
}
#endif
static rt_uint32_t spixfer(struct rt_spi_device *device, struct rt_spi_message *message)
{
    rt_uint32_t state;
    rt_size_t message_length;
    rt_uint8_t *recv_buf;
    const rt_uint8_t *send_buf;
    //rt_uint8_t pin = rt_pin_get("PE.6");
    RT_ASSERT(device != RT_NULL);RT_ASSERT(device->bus != RT_NULL);RT_ASSERT(device->bus->parent.user_data != RT_NULL);RT_ASSERT(messag
    struct ab32_soft_spi *spi_drv = rt_container_of(device->bus, struct ab32_soft_spi, spi_bus);
    struct ab32_soft_spi_pin *cs = device->parent.user_data;
    if (message->cs_take)
    {
        rt_pin_write(cs->GPIO_Pin, PIN_LOW);
    }LOG_D("%s transfer prepare and start", spi_drv->config->bus_name);LOG_D("%s sendbuf: %02x, recvbuf: %02x, length: %d",
            spi_drv->config->bus_name,
            (message->send_buf),
            ((rt_uint8_t *)(message->recv_buf)), message->length);
    message_length = message->length;
    recv_buf = message->recv_buf;
    send_buf = message->send_buf;
    if (message_length)
    {
        if (message->send_buf && message->recv_buf)
        {
            state = soft_spi_read_write_bytes(spi_drv, (rt_uint8_t *) send_buf, (rt_uint8_t *) recv_buf,
                    message_length);
            LOG_D("soft_spi_read_write_bytes");
        }
        else if (message->send_buf)
        {
            state = soft_spi_write_bytes(spi_drv, (rt_uint8_t *) send_buf, message_length);
            LOG_D("soft_spi_write_bytes");
        }
        else
        {
            memset((rt_uint8_t *) recv_buf, 0x00, message_length);
            state = soft_spi_read_bytes(spi_drv, (rt_uint8_t *) recv_buf, message_length);
            LOG_D("soft_spi_read_bytes");
        }
        if (state != message_length)
        {
            LOG_I("spi transfer error : %d", state);
            message->length = 0;
        }
        else
        {
            LOG_D("%s transfer done", spi_drv->config->bus_name);
        }
    }
    if (message->cs_release)
    {
        rt_pin_write(cs->GPIO_Pin, PIN_HIGH);
    }
    return message->length;
}
static rt_err_t spi_configure(struct rt_spi_device *device, struct rt_spi_configuration *configuration)
{
    RT_ASSERT(device != RT_NULL);RT_ASSERT(configuration != RT_NULL);
    struct ab32_soft_spi *spi_drv = rt_container_of(device->bus, struct ab32_soft_spi, spi_bus);
    spi_drv->cfg = configuration;
    return ab32_spi_init(spi_drv, configuration);
}
static const struct rt_spi_ops ab32_spi_ops = { .configure = spi_configure, .xfer = spixfer, };
static int rt_soft_spi_bus_init(void)
{
    rt_err_t result;
    for (int i = 0; i < sizeof(soft_spi_config) / sizeof(soft_spi_config[0]); i++)
    {
        soft_spi_bus_obj[i].config = &soft_spi_config[i];
        soft_spi_bus_obj[i].spi_bus.parent.user_data = &soft_spi_config[i];
        result = rt_spi_bus_register(&soft_spi_bus_obj[i].spi_bus, soft_spi_config[i].bus_name, &ab32_spi_ops);
        RT_ASSERT(result == RT_EOK);LOG_D("%s bus init done", soft_spi_config[i].bus_name);
    }
    return result;
}
/**
 * Attach the spi device to SPI bus, this function must be used after initialization.
 */
rt_err_t rt_soft_spi_device_attach(const char *bus_name, const char *device_name, hal_sfr_t cs_gpiox,
        rt_uint8_t cs_gpio_pin)
{
    RT_ASSERT(bus_name != RT_NULL);RT_ASSERT(device_name != RT_NULL);
    rt_err_t result;
    struct rt_spi_device *spi_device;
    struct ab32_soft_spi_pin *cs_pin;
    /* attach the device to spi bus*/
    spi_device = (struct rt_spi_device *) rt_malloc(sizeof(struct rt_spi_device));
    RT_ASSERT(spi_device != RT_NULL);
    cs_pin = (struct ab32_soft_spi_pin *) rt_malloc(sizeof(struct ab32_soft_spi_pin));
    RT_ASSERT(cs_pin != RT_NULL);
    cs_pin->GPIOx = cs_gpiox;
    cs_pin->GPIO_Pin = cs_gpio_pin;
    rt_pin_mode(cs_pin->GPIO_Pin, PIN_MODE_OUTPUT);
    result = rt_spi_bus_attach_device(spi_device, device_name, bus_name, (void *) cs_pin);
    if (result != RT_EOK)
    {
        LOG_E("%s attach to %s faild, %d\n", device_name, bus_name, result);
    }RT_ASSERT(result == RT_EOK);LOG_D("%s attach to %s done", device_name, bus_name);
    return result;
}
int rt_soft_spi_init(void)
{
    return rt_soft_spi_bus_init();
}
INIT_BOARD_EXPORT(rt_soft_spi_init);
#endif
#endif /* RT_USING_SPI */
```

重新挂载mount df 等操作OK

- 查看disk空间命令 `df` 查看挂载情况 `mount`

```
msh />mount sf_cmd / elm
mount device sf_cmd(elm) onto / ... succeed!
msh />mount
filesystem  device  mountpoint
----------  ------  ----------
devfs       (NULL)  /dev
elm         sf_cmd  /
msh />df
disk free: 31.9 MB [ 8182 block, 4096 bytes per block ]
msh />
```

控制台msh通信波特率 1500000bps

USBHOST设备使用

确定暂时不支持USB和蓝牙，参考

【中科蓝讯AB32VG1 RISC-V评估板】遇到的问题：USB和蓝牙 - 国产芯片交流 - 电子工程世界-论坛

【中科蓝讯AB32VG1 RISC-V评估板】遇到的问题：USB和蓝牙 ,电子工程世界-论坛

http://bbs.eeworld.com.cn/thread-1177024-1-1.html

RT-Thread-【中科蓝汛AB32VG1】开发板是否支持USB HOSTRT-Thread问答社区 - RT-Thread

RT-Thread-请问下，中科蓝汛AB32VG1是否支持USB HOST？是否有测试demo？

https://club.rt-thread.org/ask/question/431895.html

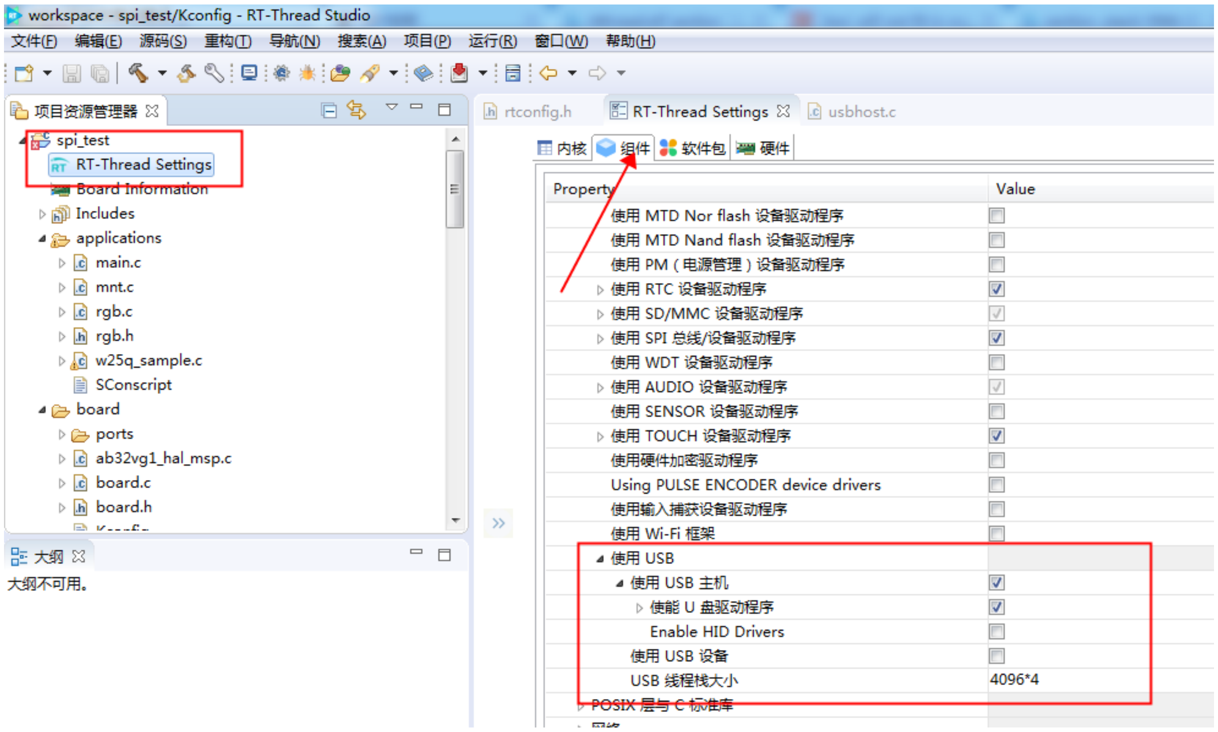
RT-Thread-有基于RT-Thread 和 AB32VG1 的蓝牙示例吗？RT-Thread问答社区 - RT-Thread

RT-Thread-搜了一下，好像没有看到蓝牙的示例。虽然没有开源，但基于lib的示例也没有。这可是主打蓝牙功能的芯片啊。

https://club.rt-thread.org/ask/question/431936.html

开启配置usb host

- 配置截图参考



Property	Value
使用 MTD Nor flash 设备驱动程序	☐
使用 MTD Nand flash 设备驱动程序	☐
使用 PM（电源管理）设备驱动程序	☐
使用 RTC 设备驱动程序	✓
使用 SD/MMC 设备驱动程序	✓
使用 SPI 总线/设备驱动程序	✓
使用 WDT 设备驱动程序	☐
使用 AUDIO 设备驱动程序	✓
使用 SENSOR 设备驱动程序	☐
使用 TOUCH 设备驱动程序	✓
使用硬件加密驱动程序	☐
Using PULSE ENCODER device drivers	☐
使用输入捕获设备驱动程序	☐
使用 Wi-Fi 框架	☐
使用 USB	
使用 USB 主机	✓
使能 U 盘驱动程序	✓
Enable HID Drivers	☐
使用 USB 设备	☐
USB 线程栈大小	4096*4
POSIX 层与 C 标准库	

- 编译出现，发现stack不够

  `ld.exe: rtthread.elf section `.bss' will not fit in region `data'`

  `ld.exe: section .stack VMA [xxx]  overlaps section .bss VMA` [xxx]

  `ld.exe: region `data' overflowed by 3296 bytes`

  参考问题输出

```
linking...
d:/rt-threadstudio/repo/extract/toolchain_support_packages/risc-v/risc-v-gcc/10.1.0/bin/../lib/gcc/riscv64-unknown-elf/10.1.0/../../../
d:/rt-threadstudio/repo/extract/toolchain_support_packages/risc-v/risc-v-gcc/10.1.0/bin/../lib/gcc/riscv64-unknown-elf/10.1.0/../../../
d:/rt-threadstudio/repo/extract/toolchain_support_packages/risc-v/risc-v-gcc/10.1.0/bin/../lib/gcc/riscv64-unknown-elf/10.1.0/../../../
collect2.exe: error: ld returned 1 exit status
make: *** [makefile:75: rtthread.elf] Error 1
"make -j4 all2" terminated with exit code 2. Build might be incomplete.
```

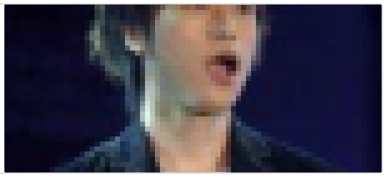
- 解解办法

- 参考链接

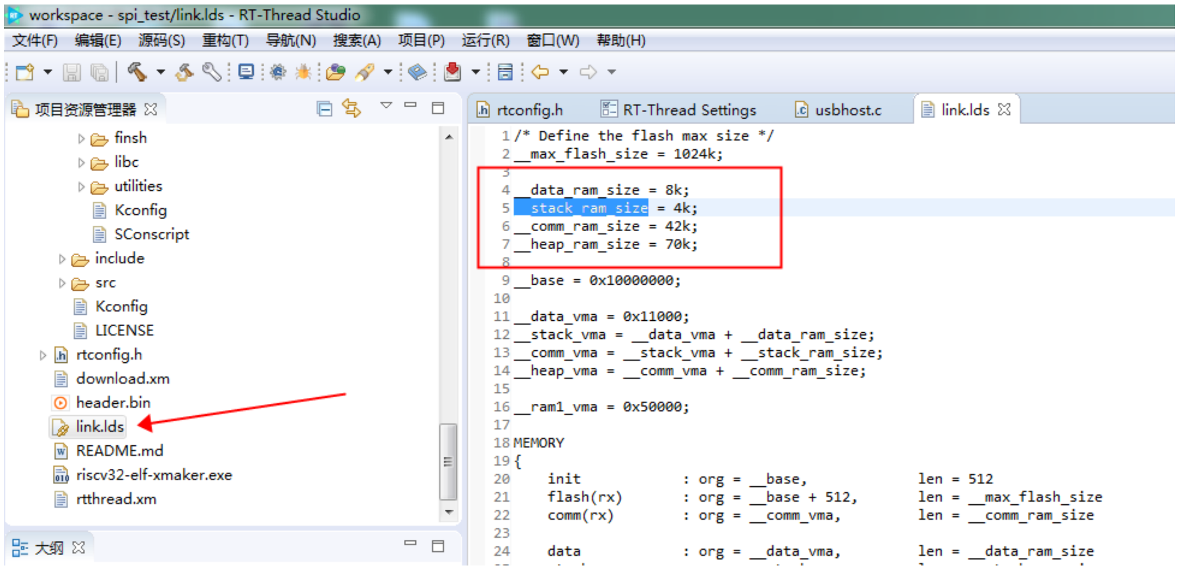
AB32VG1 RT-THREAD开发板，编译后出现内存不足的问题_西风V的博客-CSDN博客

主要如下3个问题： `.bss' will not fit in region `data' stack VMA [0000000000013000,0000000000013fff] overlaps section .bss VMA [0000000000011000,00000000000135ab] region `data' overflowed by 1452 bytes 判断是内存超出？ 这个芯片有8M的flash，192K的ram，容量足够，查询工程中涉及内存块定义的文

https://blog.csdn.net/baidu_38839743/article/details/120422801?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EOPENSEARCH%7Edefault-12.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EOPENSEARCH%7Edefault-12.no_search_link



- 操作
- 修改 `link.lds`
  - 路径图片



```
/* Define the flash max size */
__max_flash_size = 1024k;

__data_ram_size = 8k;
__stack_ram_size = 4k;
__comm_ram_size = 42k;
__heap_ram_size = 70k;

__base = 0x10000000;

__data_vma = 0x11000;
__stack_vma = __data_vma + __data_ram_size;
__comm_vma = __stack_vma + __stack_ram_size;
__heap_vma = __comm_vma + __comm_ram_size;

__ram1_vma = 0x50000;

MEMORY
{
    init            : org = __base,          len = 512
    flash(rx)       : org = __base + 512,    len = __max_flash_size
    comm(rx)        : org = __comm_vma,      len = __comm_ram_size

    data            : org = __data_vma,      len = __data_ram_size
```

  - 修改代码

```
/*
__data_ram_size = 8k;
__stack_ram_size = 4k;
__comm_ram_size = 42k;
__heap_ram_size = 70k;
*/
__data_ram_size = 16k;        /*改为16K*/
__stack_ram_size = 8k;        /*改为8K*/
__comm_ram_size = 36k;
__heap_ram_size = 64k;
```

  - 编译通过

```
make -j4 all2
linking...
riscv64-unknown-elf-objcopy -O binary "rtthread.elf"  "rtthread.bin"
riscv64-unknown-elf-objdump --source --all-headers --demangle --line-numbers --wide "rtthread.elf" > "rtthread.lst"
riscv64-unknown-elf-size --format=berkeley "rtthread.elf"
   text    data     bss     dec     hex filename
 260896       0   85216  346112   54800 rtthread.elf
sh ../pre_build.sh
riscv32-elf-xmaker -b rtthread.xm
CODE SIZE: 265 KB
save file "rtthread.dcf" successful
riscv32-elf-xmaker -b download.xm
```

- 插入U盘看看，没反应